A Memory-Hard Proof-of-Work Cryptocurrency

Version 1.0 · June 2026

Version 1.0 · June 2026

Mraksoll — Lead Maintainer
contact@bitwebcore.net
https://bitwebcore.net

Abstract

Bitweb Core (BTE) is an open-source cryptocurrency derived from Bitcoin Core 0.30.x, distinguished by three design decisions: replacement of SHA256d proof-of-work with Argon2id — a memory-hard, ASIC-resistant hashing function — adoption of the LWMA-3 per-block difficulty algorithm with a five-minute target, and a snapshot-based genesis airdrop that carried over balances from the prior chain at a 100:1 denomination. The codebase follows Bitcoin Core's development process including the full functional test suite, reproducible GNU Guix builds, and GPG-signed release attestations. This paper describes the technical parameters, the rationale behind each design choice, and the toolchain used to bootstrap the network.

Table of Contents

---

1. Introduction

Bitcoin's proof-of-work algorithm (SHA256d) was initially mined on consumer CPUs, then GPUs, and eventually displaced entirely by application-specific integrated circuits (ASICs). ASIC manufacturing concentrates hash rate among a small number of hardware producers and large mining operations, reducing the decentralization that proof-of-work is intended to enforce.

Argon2id, winner of the Password Hashing Competition (2015), was designed to be simultaneously time-hard and memory-hard. Its memory requirement makes it resistant to the parallel architectures that give ASICs their advantage over general-purpose hardware, and its sequential memory access pattern penalises GPU implementations relative to CPU ones.

Bitweb Core applies Argon2id as the proof-of-work challenge function while retaining SHA256d for all internal chain data structures — the same separation used by Litecoin (Scrypt PoW / SHA256d chain). This preserves full compatibility with Bitcoin's UTXO model, transaction format, and wallet infrastructure while making block production accessible to commodity hardware.

The network is a full relaunch of a prior Bitweb chain. Holders of the old network received new BTE proportional to their previous balance at a 100:1 denomination ratio. The relaunch was motivated by the need to upgrade the codebase to Bitcoin Core 0.30.x, adopt Argon2id, and establish cleaner monetary parameters.

---

2. Codebase and Development Process

2.1 Upstream Base

Bitweb Core is a fork of Bitcoin Core 0.30.x and tracks upstream releases. Protocol changes, wallet improvements, and security fixes from Bitcoin Core are reviewed and merged as they are published. The project commits to staying within two major version series of the Bitcoin Core upstream.

2.2 Development Standards

The development process mirrors Bitcoin Core:

- **Peer review.** All changes are submitted as GitHub pull requests and require review before merging. There is no privileged developer class; contributors earn trust through merit.
- **Functional test suite.** The full Bitcoin Core functional test suite is inherited and extended. Consensus-critical changes require passing all tests, including the regression test framework (`regtest` chain).
- **Reproducible builds.** Release binaries are built using GNU Guix for deterministic output. Builder attestations and signing keys are published at github.com/bitweb-project/guix.sigs.
- **GPG-signed releases.** Every release is signed by the lead maintainer's GPG key (`23B3 D882 F805 A5D1 F0A1 2A25 35A7 538B 1C81 6E49`). Users are expected to verify signatures before running any binary.
- **Security policy.** Vulnerabilities are reported to security@bitwebcore.net under responsible disclosure. The project maintains security updates for the two most recent major release series, following the Bitcoin Core security policy model.

---

3. Proof-of-Work: Argon2id

3.1 Algorithm Selection

Argon2 is defined in RFC 9106 and exists in three variants:

Variant	Parallelism	Data-dependence	Recommended use
Argon2d	Single-pass	Data-dependent	Cryptocurrency PoW (original)
Argon2i	Single-pass	Data-independent	Password hashing
Argon2id	**Hybrid**	**Both**	**General-purpose — RFC recommendation**

Argon2id was chosen over Argon2d since it has wide support on all platforms and programming languages in the form of ready-made libraries, unlike the I and D variants.

3.2 Consensus Parameters

The following parameters are consensus-critical and are encoded directly in `block.cpp`. Any change requires a hard fork.

Parameter	Value	Notes
Variant	`Argon2_id`	
Time cost (`t`)	`3`	Number of passes over memory
Memory cost (`m`)	`1024 KiB`	Fits in CPU L2/L3 cache; penalises GPU
Parallelism (`p`)	`1`	Single-threaded per hash attempt
Output length	`32 bytes`	256-bit PoW hash
Password	80-byte serialized header	

Salt	80-byte serialized header	Same as password

Using the serialized block header as both the password and the salt keeps the construction self-contained: the salt is fully determined by the header itself, requiring no additional hash or preprocessing step.

The 80-byte header satisfies Argon2's minimum salt requirement of 8 bytes with ample margin. The 80-byte length is identical to Bitcoin's block header serialization (4 version + 32 prevHash + 32 merkleRoot + 4 time + 4 nBits + 4 nNonce).

3.3 Hash Construction

```
// From block.cpp — consensus-critical, must not be changed
argon2_context context;
context.out     = hash.begin();
context.outlen  = 32;
context.pwd     = (uint8_t*)ss.data();   // 80-byte serialized header
context.pwdlen  = 80;
context.salt    = (uint8_t*)ss.data();   // same as password
context.saltlen = 80;
context.t_cost  = 3;
context.m_cost  = 1024;
context.lanes   = 1;
context.threads = 1;
context.version = ARGON2_VERSION_NUMBER;

const int rc = argon2_ctx(&context, Argon2_id);
assert(rc == ARGON2_OK);
```

3.4 Chain Hash Separation

Block identifiers (used in `hashPrevBlock`, Merkle tree roots, and all chain linkage) are computed with SHA256d, not Argon2id. Argon2id serves exclusively as the proof-of-work challenge. This is the same separation used by Litecoin (Scrypt PoW / SHA256d chain data) and ensures that:

- Chain indexing and validation remain computationally cheap.
- Existing block explorer and wallet infrastructure is compatible without modification.
- The PoW challenge can in principle be changed in a future hard fork without altering chain data structures.

3.5 Memory Hardness and ASIC Resistance

At `m = 1024 KiB`, each hash attempt requires filling and reading one mebibyte of working memory. Because Argon2's internal Blake2b rounds create data dependencies between blocks, the memory cannot be computed in parallel sub-units — it must be traversed sequentially. This eliminates the area reuse that makes SHA256d ASICs economically viable: an ASIC targeting Argon2id at these parameters requires the same ~1 MiB SRAM per hashing core as a general-purpose CPU, removing the area advantage that ASICs ordinarily exploit.

At `t = 3` passes, three sequential sweeps over the 1 MiB buffer are required per hash. This increases the time cost relative to a single pass without increasing the peak memory footprint, further raising the cost of time-memory trade-off (TMTO) attacks.

---

4. Implementation: ISA Dispatcher

The internal fill-segment function — the innermost loop of Argon2's memory-filling phase — is performance-critical. Bitweb Core implements the same ISA-dispatch pattern used by Bitcoin Core for SHA256: a runtime CPU feature detector selects the fastest available implementation.

4.1 Available Backends

Backend	Compiled with	Condition
`fill_segment_ref`	(always)	Pure-C reference; safe fallback
`fill_segment_sse2`	`-msse2`	x86: SSE2 detected at runtime
`fill_segment_ssse3`	`-mssse3`	x86: SSSE3 detected at runtime
`fill_segment_avx2`	`-mavx2`	x86: AVX2 + OS XSAVE enabled
`fill_segment_avx512`	`-mavx512f`	x86: AVX-512F + OS XSAVE enabled
`fill_segment_neon`	`-mfpu=neon`	AArch64: NEON flag in `use_implementation`

4.2 Dispatcher Mechanism

A global function pointer `argon2_fill_segment` defaults to `fill_segment_ref` at startup. `Argon2AutoDetectImpl()` is called once at node startup. It performs CPUID detection and, on x86, verifies OS XSAVE support before upgrading the pointer to the fastest available SIMD backend. On AArch64, NEON is selected when available.

This design means every node automatically uses the most efficient implementation available on its hardware without requiring separate binary distributions per CPU microarchitecture.

---

5. Difficulty Algorithm: LWMA-3

5.1 Rationale for Replacing Bitcoin's Retargeting

Bitcoin adjusts difficulty once every 2,016 blocks (approximately two weeks). This large window was appropriate for a network with stable hash rate, but creates significant instability when hash rate changes rapidly — a common occurrence for smaller networks where a single pool can represent a significant fraction of total hash rate. A 2,016-block window also means the network is effectively unable to respond to hash rate changes for up to two weeks.

Bitweb Core uses LWMA-3 (Linearly Weighted Moving Average, revision 3), designed by Zawy with contributions from the Bitcoin Gold and Zcash communities. LWMA-3 retargets on every block, using a weighted average of the N most recent solve times where more recent blocks receive linearly higher weight. This allows the network to track hash rate changes within hours rather than weeks.

5.2 Parameters

Parameter	Value	Notes
Target block time `T`	`300 s` (5 min)	Between LTC (150 s) and BTC (600 s)
Averaging window `N`	`576 blocks`	
Retarget	Every block	

The 5-minute block time was chosen to place Bitweb between Litecoin's 2.5 minutes and Bitcoin's 10 minutes. Confirmation latency is half that of Bitcoin while the longer interval reduces orphan rate on variable-latency network paths.

5.3 Genesis Mining

The first 60,000 blocks were mined at minimum difficulty (`powLimit`) for genesis airdrop distribution to prior chain holders. LWMA-3's normal operation begins at block 60,001.

---

6. Timing and Anti-Manipulation Parameters

6.1 Future Time Limit (FTL)

```
// chain.h
static constexpr int64_t MAX_FUTURE_BLOCK_TIME = 600; // 2 × target block time
```

A block is rejected if its timestamp exceeds the node's current time by more than 600 seconds. Setting FTL to `2T` (two block intervals) is standard practice for time-warp attack mitigation. The original Bitcoin Core value was `7200 s` (two hours); Bitweb Core reduces this to limit the window in which a miner can manipulate timestamps to artificially lower difficulty.

6.2 Timestamp Window

```
static constexpr int64_t TIMESTAMP_WINDOW = 2 * 60 * 60; // 2 hours
```

The grace period used when validating external timestamps (RPC parameters, wallet key creation times) against block timestamps. This remains at two hours to preserve compatibility with wallet software that computes key creation time from block timestamps.

6.3 Catching-Up Gap

```
        static constexpr int64_t MAX_BLOCK_TIME_GAP = 90 * 60; // 90 minutes
```

The threshold at which the GUI switches to "Catching up..." mode. At a 5-minute block time, 90 minutes represents 18 blocks — enough to distinguish a genuine sync lag from normal network variance.

6.4 Summary

Constant	Value	Ratio to T
`MAX_FUTURE_BLOCK_TIME`	600 s	2T
`TIMESTAMP_WINDOW`	7200 s	24T
`MAX_BLOCK_TIME_GAP`	5400 s	18T

7. Monetary Policy

7.1 Supply Cap

The maximum supply is capped at `42,000,000 BTE` in the consensus parameters. The subsidy function uses integer right-shift identical to Bitcoin Core's `GetBlockSubsidy()` implementation. Actual issuable supply across all 33 halving eras is **41,999,999.9538 BTE** (4,199,999,995,380,000 satoshis — the C++ verifiable nSum value).

7.2 Block Reward and Halving

```
Initial block reward : 50 BTE  (5,000,000,000 satoshis)
Halving interval     : 420,000 blocks
Halving period       : ≈ 4.0 years at 300 s/block
```

The reward for block `n` in satoshis:

```
subsidy(n) = (50 × COIN) >> floor(n / 420000)
COIN = 100,000,000
```

This mirrors Bitcoin Core exactly. The right-shift produces precise fractional BTE values (12.5, 6.25, 3.125…) unlike floor division of integer BTE amounts.

7.3 Halving Schedule

Era	Blocks	Reward (BTE)	Era Supply (BTE)	Cumulative (BTE)	Approx. Year
0	0 – 419,999	50	21,000,000	21,000,000	2026–2030
1	420,000 – 839,999	25	10,500,000	31,500,000	2030–2034
2	840,000 – 1,259,999	12.5	5,250,000	36,750,000	2034–2038
3	1,260,000 – 1,679,999	6.25	2,625,000	39,375,000	2038–2042

4	1,680,000 – 2,099,999	3.125	1,312,500	40,687,500	2042–2046
5	2,100,000 – 2,519,999	1.5625	656,250	41,343,750	2046–2050

Series continues for 27 additional eras. Total converges to **41,999,999.9538 BTE** (verifiable nSum: 4,199,999,995,380,000 satoshis).

7.4 Coinbase Maturity

A standard coinbase output becomes spendable after `100` blocks (approximately 8.3 hours). The extended maturity rule for the vesting window is described in Section 11.

---

8. Network Parameters

8.1 Ports and Magic

Parameter	Value
P2P Port (mainnet)	`26333`
RPC Port (mainnet)	`26332`
Network magic	`0xfeaed5ca`
BIP-324 shared secret salt	`bitweb_v2_shared_secret`
BIPs active from	Block 1

8.2 Address Formats

Type	Prefix	Version byte	Encoding
P2PKH	`E…`	`33` (0x21)	Base58Check
P2SH	`D…`	`30` (0x1E)	Base58Check
P2WPKH / P2WSH	`web1q…`	—	Bech32
P2TR (Taproot)	`web1p…`	—	Bech32m

All SegWit and Taproot address types inherited from Bitcoin Core are fully supported and active from block 1.

---

9. Consensus-Level Security Patches

Bitweb Core incorporates several security patches applied at the consensus layer, beyond what was present in the inherited Bitcoin Core base.

9.1 BIP53 — 64-Byte Transaction Rejection

```
// tx_check.cpp
if (!tx.IsCoinBase() && ::GetSerializeSize(TX_NO_WITNESS(tx)) == 64)
    return state.Invalid(TxValidationResult::TX_CONSENSUS, "bad-txns-64byte",
                         "tx serialized size is exactly 64 bytes (BIP53)");
```

Transactions serialized to exactly 64 bytes (without witness) are rejected at the consensus level. Such transactions create ambiguity in the Merkle tree: a 64-byte transaction serialization can be confused with an internal Merkle node, enabling certain Merkle proof forgeries. Coinbase transactions are exempt because a 64-byte coinbase would require approximately 224 bits of work to construct and is therefore practically impossible to produce maliciously.

9.2 CVE-2018-17144 — Duplicate Input Detection

Transactions are checked for duplicate inputs before any UTXO lookup. The original vulnerability could result in coin inflation if a transaction spending the same output twice passed validation in certain caching conditions.

---

10. Genesis Airdrop

10.1 Background

The prior Bitweb chain had an established holder base. To preserve holder equity across the relaunch, a full balance snapshot of the old chain was taken at its final block and used as the initial allocation for new BTE.

10.2 Snapshot Methodology

A purpose-built UTXO scanner (`btesnapshot.py`) performed a full scan of the old chain via the Bitcoin RPC `getblock` interface with verbosity level 2, processing all transaction inputs and outputs in order to reconstruct the final UTXO set. All standard script types were supported:

Script type	Address resolution
P2PKH	Decoded by node; Bitweb prefix 33 (0x21)
P2SH	Decoded by node; Bitweb prefix 30 (0x1E)
P2WPKH	Decoded by node; Bech32 `web1q…`
P2WSH	Decoded by node; Bech32 `web1q…`
P2TR	Decoded by node; Bech32m `web1p…`
P2PK	Derived manually: Hash160(pubkey) → Base58Check with version byte 33
`OP_RETURN`	Skipped (unspendable by design)

The scanner operated in batches of 1,000 blocks per RPC round-trip, with a two-phase design: a validation run over the first 1,000 blocks, followed by a full scan upon manual

confirmation.

The output was a file of `address:satoshis` pairs representing the final UTXO-aggregated balance of every address that held coins at snapshot height.

10.3 Denomination

A 100:1 denomination was applied: one new BTE for every 100 old coins. This was implemented with ceiling rounding to avoid penalising small holders:

```
        # denominator.py
        sat_new = math.ceil(sat_old / 100)
```

Any address whose denominated balance fell below `10,000 satoshis` was raised to `10,000 satoshis` (the dust floor). This ensures every holder receives a balance above the standard relay dust threshold, regardless of how small their position on the old chain was.

10.4 Distribution Mechanics

The denominated airdrop file was submitted to the new chain using `sendmany` RPC calls in batches of 500 addresses, with a fee rate of `2 sat/vB` and a checkpoint-resume mechanism so that a interrupted airdrop process could continue from the last confirmed batch without re-sending any payment.

Each batch was confirmed before the next was submitted, enforcing sequential ordering and preventing fee market congestion from simultaneous large-scale output creation.

---

11. Vesting and Anti-Dump Protection

11.1 Rationale

A snapshot-based airdrop creates an immediate supply overhang: all distributed coins are simultaneously unencumbered at genesis. Without a mitigation, large holders from the old chain could sell their entire allocation in the first blocks, suppressing price discovery and disadvantaging new participants who discover the network later.

11.2 Mechanism

Bitweb Core introduces an extended coinbase maturity rule applied to blocks 60,000 through 69,299. These blocks were mined during the distribution window immediately following the genesis airdrop.

```
        // Consensus::CheckTxInputs (tx_check excerpt)
        static constexpr int EXT_MATURITY_START = 60000;
        static constexpr int EXT_MATURITY_END   = EXT_MATURITY_START + EXT_COINBASE_MATURITY; //
69300

        if (coin.nHeight >= EXT_MATURITY_START && coin.nHeight < EXT_MATURITY_END) {
```

```
            // Required depth = (EXT_MATURITY_END - coin.nHeight) + COINBASE_MATURITY
            if (nSpendHeight - coin.nHeight < (EXT_MATURITY_END - coin.nHeight) + COINBASE_MATUR
                return state.Invalid(..., "bad-txns-premature-spend-of-coinbase", ...);
        }
```

The maturity requirement is computed such that for any block `h` in `[60000, 69299]`, the coinbase output becomes spendable at block `69300 + 100 = 69400`. Every coinbase from the extended window unlocks simultaneously at block **69,400**.

11.3 Key Blocks

Event	Block
Pre Miner for airdrop/swap distribution	0 – 59,999
Extended maturity window starts	60,000
Extended maturity window ends	69,299
All extended coinbases unlock	69,400

11.4 Properties

- **Cliff, not gradual release.** All locked coins unlock at the same block (69,400). This is a deliberate design choice: gradual release schedules create recurring sell pressure events. A single cliff gives the market a known, predictable date to price in.
- **Miner reward unaffected.** The block reward is always 50 BTE throughout this period. Only spendability is delayed, not the reward itself.
- **Standard maturity resumes at 69,400.** After the cliff, all coinbases mature after the standard 100 blocks (~8.3 hours).

11.5 Pool Operator Note

A pool accumulating one coinbase per block across the 9,300-block window will hold over 1,600 unspent coinbase outputs by the time it attempts a large payout. A `sendmany` transaction spending more than ~500 of these inputs may exceed the standard transaction size limit. Pool operators should consolidate in batches of no more than 50,000 BTE per transaction, and perform consolidation after block 69,400 to avoid extended-maturity errors.

---

13. Conclusion

Bitweb Core applies three targeted modifications to Bitcoin Core: Argon2id as a memory-hard proof-of-work function, LWMA-3 per-block difficulty retargeting calibrated to a five-minute block interval, and a snapshot-based genesis airdrop with a vesting cliff to manage initial supply distribution. All other Bitcoin Core behaviour — transaction format, UTXO model, SegWit, Taproot, BIP-324 transport encryption, reproducible builds, and

the full test suite — is preserved.

The design prioritises verifiability. Every consensus-critical parameter is embedded directly in source code and documented here. Every release binary is independently reproducible via GNU Guix and GPG-signed. The development process is open: there is no privileged developer class, and all changes go through public peer review.

Source: github.com/bitweb-project/bitweb
Website: bitwebcore.net

Appendix A — Network Parameter Reference

Parameter	Value
Ticker	BTE
Max supply (params)	42,000,000 BTE
Issuable supply (satoshi precision)	41,999,999.9538 BTE
nSum (C++ verifiable)	4,199,999,995,380,000 sat
Initial block reward	50 BTE
Halving interval	420,000 blocks
Halving period	≈ 4.0 years
Target block time	300 s (5 min)
Coinbase maturity (standard)	100 blocks (~8.3 h)
Coinbase maturity (extended)	Variable; all unlock at block 69,400
PoW algorithm	Argon2id
Argon2id `t`	3
Argon2id `m`	1024 KiB
Argon2id `p`	1
Difficulty algorithm	LWMA-3
LWMA window N	576 blocks
Retarget	Every block
Max future block time (FTL)	600 s
Pre - Mined blocks for Airdrop/Swap and project needs.	60,000
PoW limit	`000fffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff` or `0x1f0fffff`
Genesis block hash	`111692c1b9b390c407ab74d7f924d4fa0f7589974ab61af96392feca11f209e6`
P2P port (mainnet)	26333

RPC port (mainnet)	26332
Network magic	0xfeaed5ca
BIP-324 salt	`bitweb_v2_shared_secret`
P2PKH prefix	E (dec 33, 0x21)
P2SH prefix	D (dec 30, 0x1E)
Bech32 HRP	`web`
SegWit / Taproot	Active from block 1
BIPs	Active from block 1
License	MIT

Appendix B — Airdrop Toolchain

Three scripts were used in sequence to produce and distribute the genesis airdrop:

1. `btesnapshot.py` — UTXO scanner.
Connects to the old chain's RPC node, scans all blocks in batches of 1,000, reconstructs the final UTXO set, resolves addresses for all standard script types including P2PK (manual derivation with Bitweb version byte 33), and outputs `address:satoshis` pairs. can be found at https://github.com/bitweb-project/bitweb-maintainer-tools/blob/main/btesnapshot.py

2. `denominator.py` — Denomination and dust adjustment.
Reads the snapshot file, applies 100:1 denomination with ceiling rounding, raises any result below 10,000 satoshis to 10,000 satoshis (dust floor), and outputs the adjusted `address:satoshis` file sorted descending by balance. can be found at https://github.com/bitweb-project/bitweb-maintainer-tools/blob/main/denominator.py

3. `aitdrop.py` — Distribution sender.
Reads the denominated file, batches addresses into groups of 500, and submits each batch via `sendmany` RPC with fee rate 2 sat/vB. Maintains a checkpoint file so interrupted runs resume from the last confirmed batch. Waits for at least one confirmation per batch before proceeding. can be found at https://github.com/bitweb-project/bitweb-maintainer-tools/blob/main/aitdrop.py

Appendix C — Halving Schedule

Era	Block range	Reward (BTE)	Era supply (BTE)	Cumulative (BTE)	Target year
0	0 – 419,999	50	21,000,000	21,000,000	2026–2030
1	420,000 – 839,999	25	10,500,000	31,500,000	2030–2034
2	840,000 – 1,259,999	12.5	5,250,000	36,750,000	2034–2038

3	1,260,000 – 1,679,999	6.25	2,625,000	39,375,000	2038–2042
4	1,680,000 – 2,099,999	3.125	1,312,500	40,687,500	2042–2046
5	2,100,000 – 2,519,999	1.5625	656,250	41,343,750	2046–2050
6	2,520,000 – 2,939,999	0.78125	328,125	41,671,875	2050–2054
7	2,940,000 – 3,359,999	0.390625	164,062.5	41,835,937.5	2054–2058
…	…	…	…	…	…
33	13,440,000 – 13,859,999	0.00000001	0.0042	41,999,999.9538	~2162

Reward halves every 420,000 blocks. Series runs 33 eras until reward = 1 satoshi (0.00000001 BTE). Total issuable: **41,999,999.9538 BTE** (nSum: 4,199,999,995,380,000 satoshis).